DUMPSBOSS.

# Developing Applications for Cisco Webex and Webex Devices (DEVWBX)

## Cisco 300-920

## Version Demo

**Total Demo Questions: 9**

**Total Premium Questions: 95**

**Buy Premium PDF**

https://dumpsboss.co

support@dumpsboss.co

support@dumpsboss.co
dumpsboss.co

# Topic Break Down

| Topic | No. of Questions |
|---|---|
| Topic 1, Webex platform | 2 |
| Topic 2, Webex Meetings | 14 |
| Topic 3, Webex Devices | 24 |
| Topic 4, Webex integrations | 55 |
| Total | 95 |

QUESTION NO: 1 - (DRAG DROP)

Refer to the exhibit.

A Webex device In-Room Control editor screenshot and associated Macro code is shown. Drag and drop the code snippets to complete the JavaScript Macro that launches a call when the Call button on the custom control panel is touched. Not all options are used.

```
const xapi = require('xapi');

xapi.event.on('                                    ', (event)=> {
    if (event.WidgetId !== '                                    ') return;
    if (event.Type !== 'clicked') return;
        xapi.                                    
('                                    ', (Number: '1000'});
});
```

UserInterface InRoomControl Button Event

call_1

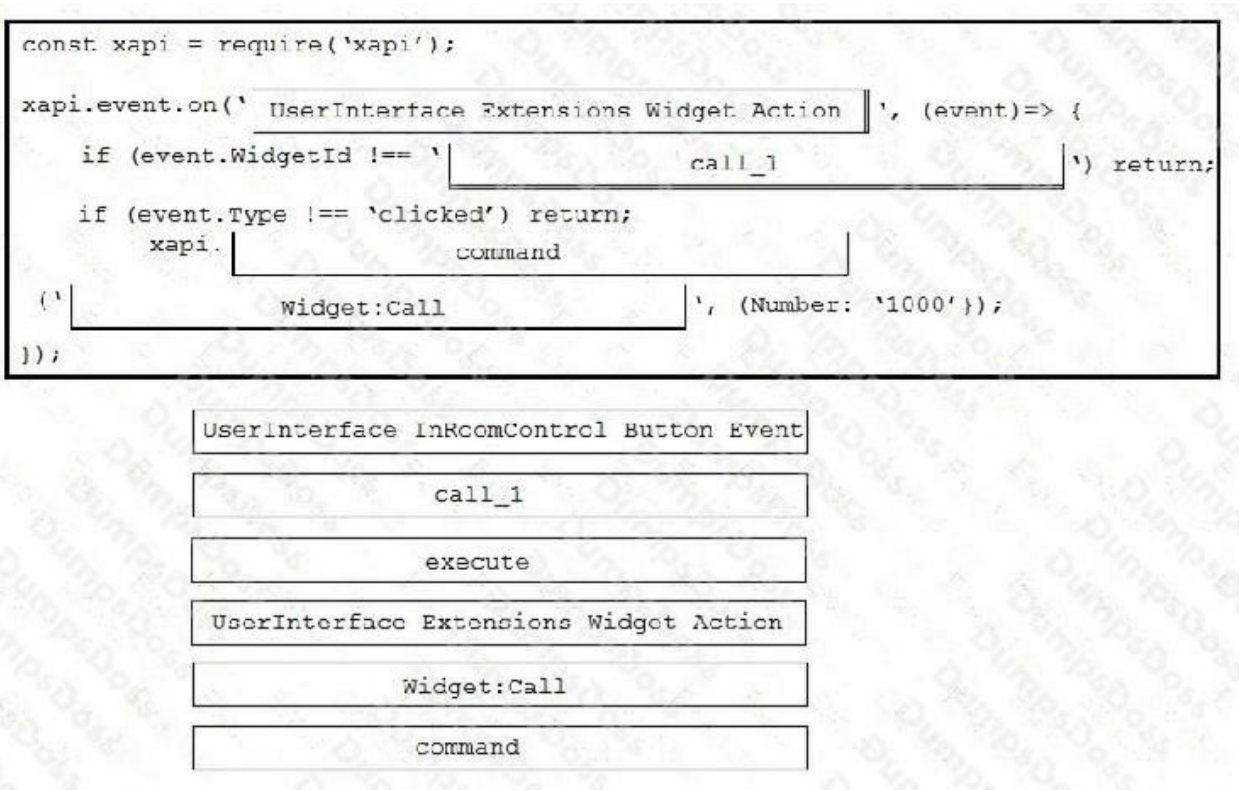
execute

UserInterface Extensions Widget Action

Widget:Call

command

**ANSWER:**




**Explanation:**

The completed macro uses the Webex device xAPI event-and-command pattern to respond to an interaction with an In-Room Control widget. The event subscription must be `UserInterface Extensions Widget Action`, because this is the event emitted when a user interacts with a custom widget created in the In-Room Control editor. The editor exhibit identifies the Call button with the widget ID `call_1`; matching that exact value in `event.WidgetId` ensures that the macro handles the intended control.

The supplied `event.Type` condition keeps the action specific to a touch/click interaction. Once both the widget ID and click event have been validated, the macro invokes the xAPI command interface through `xapi.command`. The command argument is `Widget:Call`, and the supplied object passes the destination using `{Number: '1000'}`. In full, the completed logic subscribes to the widget-action event, checks for `call_1`, verifies a click, and executes `xapi.command('Widget:Call', {Number: '1000'})`. This structure cleanly separates receiving the UI event from executing the requested device action and is the expected construction for the blanks shown in the exhibit.

Cisco documents the JavaScript xAPI module model, including event subscriptions and command execution, in its RoomOS xAPI reference. Cisco's customization guide also describes building custom user-interface controls and using macros to react to their events.

## QUESTION NO: 2

Refer to the exhibit.

```
<xsd:complexType name= "lstRecordingResponse">
     <xsd:complexContent>
          <xsd:extension base= "serv:bodyContentType">
               <xsd:sequence>
                    <xsd:element name= "matchingRecords" type=
"serv:matchingRecordsType" minOccurs="0"/>
                    <xsd:element name= "recording" type= "ep:recordingType" minOccurs=
"0" maxOccurs= "unbounded"/>
...
<xsd:complexType name= "recordingType">
     <xsd:sequence>
          <xsd:element name= "recordingID" type= "xsd:int"/>
          <xsd:element name= "hostWebExID" type= "xsd:string"/>
          <xsd:element name= "name" type= "xsd:string"/>
          <xsd:element name= "createTime" type= "xsd:string"/>
          <xsd:element name= "sreamURL" type= "xsd:string"/>
          <xsd:element name= "fileURL" type= "xsd:string"/>
          <xsd:element name= "password" type= "xsd:string" minOccurs= "0" />
```

A snippet from the XSD schema of the Webex Meeting XML API 'LstRecordingResponse' element is listed in the exhibit. Assuming that a variable named 'resp' exists that contains the XML response from a successful 'LstRecording' request, which code snippet correctly generates a simple report that lists meeting names and recording file download links?

A.
```
list = (new window.DOMParser()).parseFromString(resp, 'application/json')
for (const item of lst.getElementsByTagNameNS('*', 'matchingRecords')){
document.writeln(
 <p> Meeting Name:${item.getElementsByTagNameNS('*', 'name') [0].textContent}<br>');
document.write(
'Download:${item.getElementsByTagNameNS('*', 'fileURL') [0].textContent}<br>');
```

B.
```
list = (new window.DOMParser()).parseFromString(resp, 'LstRecordingResponse')
for (const item of lst.getElementsByTagNameNS('*', 'matchingRecords')){
document.writeln(
 <p> Meeting Name:${item.getElementsByTagNameNS('*', 'name') [0].textContent}<br>');
document.write(
'Download:${item.getElementsByTagNameNS('*', 'fileURL') [0].textContent}<br>');
```

C.
```
list = (new window.DOMParser()).parseFromString(resp, 'text/xml')
for (const item of lst.getElementsByTagNameNS('*', 'matchingRecords')){
document.writeln(
 <p> Meeting Name:${item.getElementsByTagNameNS('*', 'hostWebExID') [0] }<br>');
document.write(
'Download:${item.getElementsByTagNameNS('*', 'streamURL') [0].textContent}<br>');
```

D.
```
list = (new window.DOMParser()).parseFromString(resp, 'text/xml')
for (const item of lst.getElementsByTagNameNS('*', 'recording')){
document.writeln(
 <p> Meeting Name:${item.getElementsByTagNameNS('*', 'name') [0].textContent}<br>');
document.write(
'Download:${item.getElementsByTagNameNS('*', 'fileURL') [0].textContent}<br>');
```

**A.** Option A

**B.** Option B

**C.** Option C

**D.** Option D

**ANSWER: A**

**Explanation:**

The correct snippet traverses the repeating recording entries in the `LstRecordingResponse` XML payload and, for each entry, retrieves the meeting-name field and its associated recording download-link field. This follows the XSD structure: the

response is a container for zero or more recording elements, rather than a single recording object. Therefore, report generation must iterate through the recording collection before emitting a line containing the topic/name and file URL for each result.

It is also essential that the XML query uses the element hierarchy and namespaces defined by the Webex Meeting XML API schema. A successful `LstRecording` response can include multiple recordings, and each recording's file link is associated with that specific recording element. Keeping the field lookups scoped to the current recording prevents meeting names and links from being mismatched. The resulting output is a simple per-recording report in which each meeting is paired with its own downloadable recording resource. Cisco's Webex developer documentation provides the current recordings resource model and download-related information at Webex Recordings API documentation; the Meeting XML API reference material is available through Cisco Webex XML API Reference Guide.

## QUESTION NO: 3

Which REST API request is used to list all the Webex Room Kit devices within a large organization so that a new custom In-Room Control can be deployed on all the devices?

A.

```
var request = require("request");
var options = { method: 'GET',
        url: 'https://api.ciscospark.com/v1/devices',
        qs: { product: 'Roomkit' },
        headers:
         { 'Content Type': 'application/json',
           Authorization: 'Bearer Yz6FgoWx7Pgb57C9z' }};

request(options, function(error, reponse, body) {
        if (error) throw new Error(error);
        console.log(body);
});
```

B.

```
var request = require("request");
var options = { method: 'GET',
        url: 'https://api.ciscospark.com/v1/devices',
        qs: { product: 'Roomkit' , placeID: 'Yzb60gRx3kBq5iB2w' },
        headers:
         { 'Content Type': 'application/json',
           Authorization: 'Bearer Yz6FgoWx7Pgb57C9z' }};

request(options, function(error, reponse, body) {
        if (error) throw new Error(error);
        console.log(body);
});
```

C.

```
var request = require("request");
var options = { method: 'GET',
        url: 'https://api.ciscospark.com/v1/devices/Yzb60gRx3kBq5iB2w'
        qs: { deviceName: 'Roomkit' },
        headers:
         { 'Content Type': 'application/json',
           Authorization: 'Bearer Yz6FgoWx7Pgb57C9z' }};

request(options, function(error, reponse, body) {
        if (error) throw new Error(error);
        console.log(body);
});
```

D.

```
var request = require("request");
var options = { method: 'GET',
        url: 'https://api.ciscospark.com/v1/devices',
        qs: { upgradeChannel: 'Roomkit' },
        headers:
         { 'Content Type': 'application/json',
           Authorization: 'Bearer Yz6FgoWx7Pgb57C9z' }};

request(options, function(error, reponse, body) {
        if (error) throw new Error(error);
        console.log(body);
});
```

**A.** GET https://webexapis.com/v1/devices?product=RoomKit

**B.** Option B

**C.** Option C

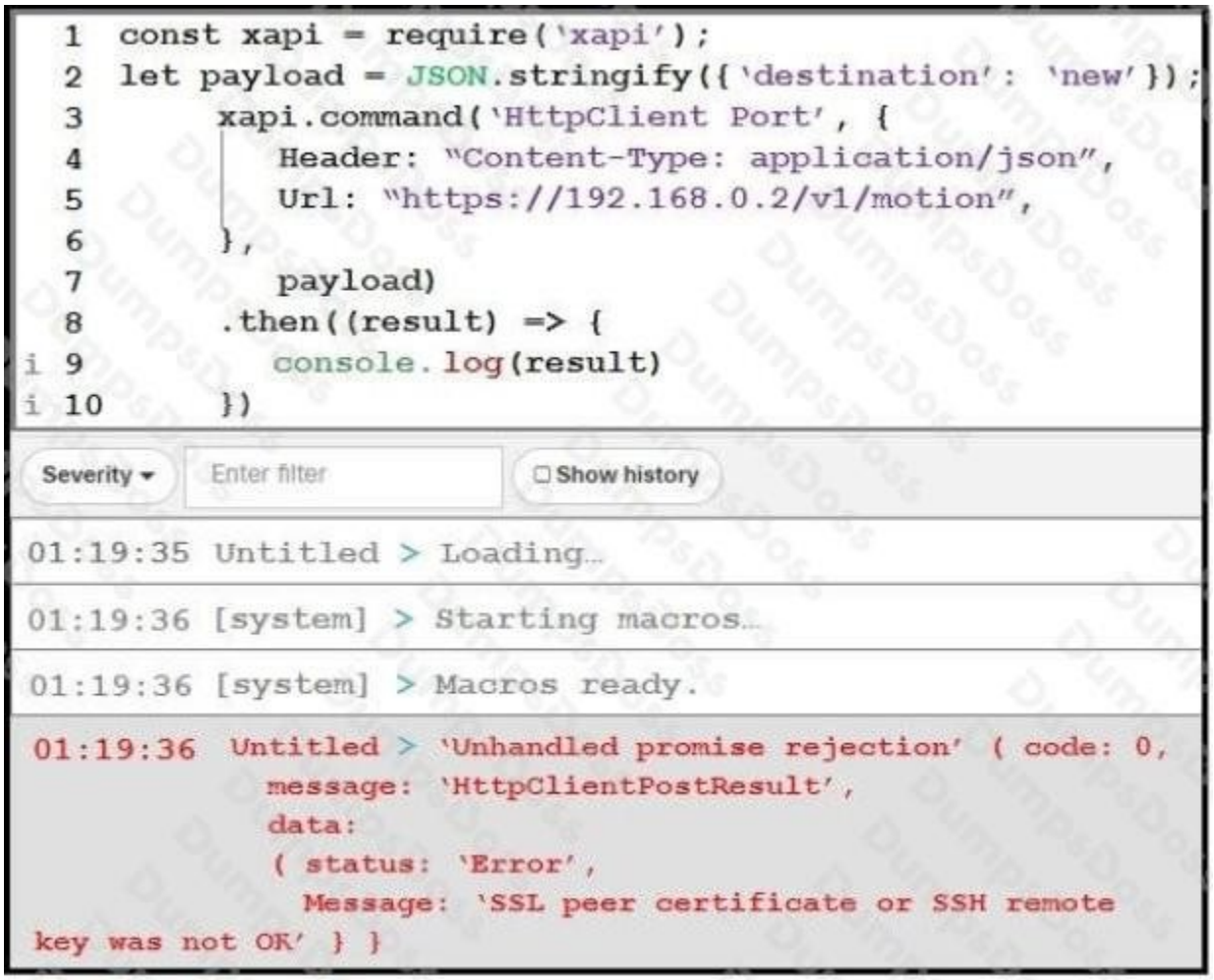
**D.** Option D

**ANSWER: A**

**Explanation:**

`GET https://webexapis.com/v1/devices?product=RoomKit` is the appropriate request because the Webex Devices API provides the `GET /v1/devices` endpoint for retrieving devices that are managed in the organization associated with the access token. Its `product` query parameter narrows the returned device inventory to the specified device product, allowing an application to identify the Room Kit estate rather than processing every registered device type in the organization. The request requires an access token with the appropriate organization-level device permissions, typically supplied as an OAuth bearer token or a suitably authorized administrator integration token. For a large organization, the application must follow the pagination information returned by the API, including the `Link` response header, until all pages of matching Room Kit devices have been retrieved. The resulting device IDs can then be used as the target inventory for the automation workflow that deploys the In-Room Control. Cisco documents the endpoint and supported filtering parameters in the List Devices API reference. The broader device-management API model is also described in the Webex Devices API guide.

**QUESTION NO: 4**




Refer to the exhibit. What causes the error message?

**A.** xapi must be enabled for promises.

**B.** HttpClient AllowInsecureHTTPS has not been enabled.

**C.** The NODE_TLS_REJECT_UNAUTHORIZED environment variable must be set to 0.

**D.** HttpClient must be changes to HttpsClient.

**ANSWER: B**

**Explanation:**

The error is caused by attempting an HTTPS request to a server whose TLS certificate cannot be validated by the Webex device, typically because it is self-signed or issued by an untrusted certificate authority. The device's HTTP client validates HTTPS certificates by default as a security control. For development or controlled environments that use a self-signed certificate, the device must explicitly be configured to permit insecure HTTPS connections by enabling `HttpClient AllowInsecureHTTPS`. This configuration allows the HTTP client to complete requests even when normal certificate verification cannot establish trust.

Webex devices expose HTTP client functionality through xAPI, including configuration controls for HTTPS behavior. Cisco's documentation notes that certificate validation is enforced unless insecure HTTPS is allowed, and that this setting should be used carefully because bypassing validation weakens protection against impersonation and man-in-the-middle attacks. In a production deployment, the preferable approach is to install a certificate issued by a trusted CA and ensure that the server name matches the certificate. For the request and error shown, however, enabling `HttpClient AllowInsecureHTTPS` is the required configuration change. See Cisco's Sending HTTP Requests from a Board, Room, or Desk Device documentation and the RoomOS xAPI reference.

## QUESTION NO: 5

```
const Webex = require('webex');

const webex = Webex.init({
    credentials: {
        access_token: 'A_VALID_ACCESS_TOKEN'
    }
});
```

Refer to the exhibit. When using the Webex Browser SDK to create calls and share screens, which two statements are valid given a 'webex' object such as displayed in the exhibit? (Choose two.)

**A.** After a meeting is joined, it cannot be left programmatically until the host ends the meeting.

**B.** The webex meetings.register() function must be invoked before attempting to join any meeting.

**C.** The joinMeeting() function throws an error of type 'media stopped' if a media stream is stopped.

**D.** Given a Webex meeting number the webex meetings join() function can be used to join the meeting.

**E.** The mediaSettings for a joined meeting accepts boolean attributes to send and receive audio, video, and screen share.

## ANSWER: B D

**Explanation:**

`webex.meetings.register()` must be completed before an application attempts to join meetings. Registration associates the Browser SDK client with the Webex meetings service and prepares the client to receive meeting-related events and signaling. In practice, applications normally register after the SDK is initialized and authenticated, then perform meeting lookup, creation, or join operations.

Given a meeting number, `webex.meetings.join()` is the SDK entry point used to obtain the meeting instance for that destination and proceed with joining it. The returned meeting object represents the meeting session and provides the meeting-level operations used by the application, including the subsequent join and media workflow. This allows a browser application to locate and enter a meeting using its meeting number rather than requiring a pre-existing meeting object.

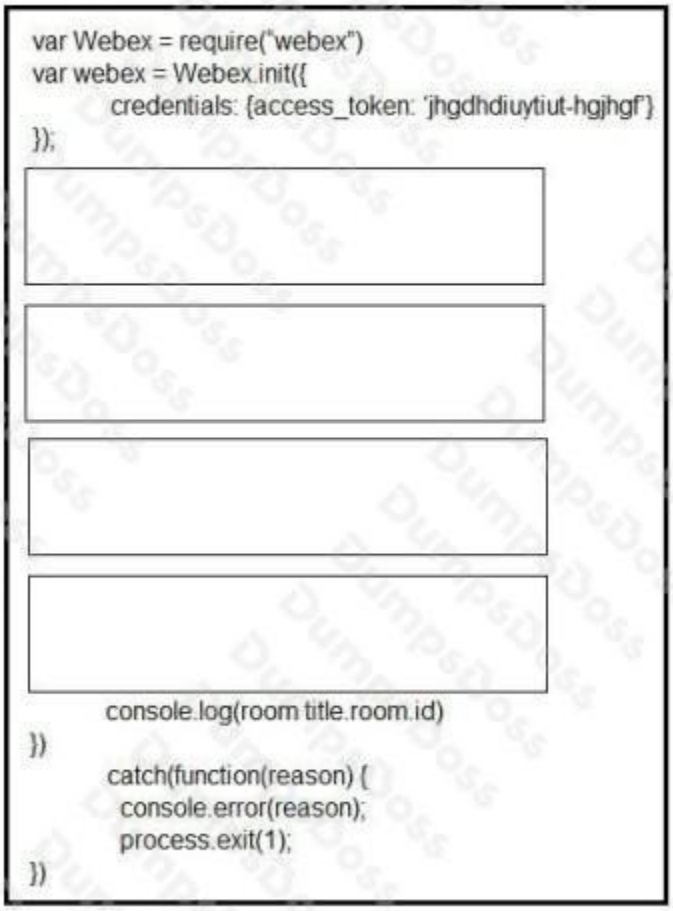
These operations are part of the Webex Browser SDK meeting workflow; the SDK documentation describes registering the meetings plugin before meeting activity and using the meetings API to join a destination. See the Webex Browser SDK documentation and the Webex JavaScript SDK API reference.

## QUESTION NO: 6 - (DRAG DROP)

DRAG DROP

Drag and drop the code segments from the right of the screen into the targets on the left to create a script that retrieves a list of rooms using the Webex JavaScript API and outputs the title and ID to the console for one that matches the title "My Webex discussion". Not all code segments are used.

**Select and Place:**




webex.rooms

.list({
max:500
})

var room = rooms.items.filter(function(room){
return title.room == "My Webex Discussion"
});

.resolve(function(rooms){

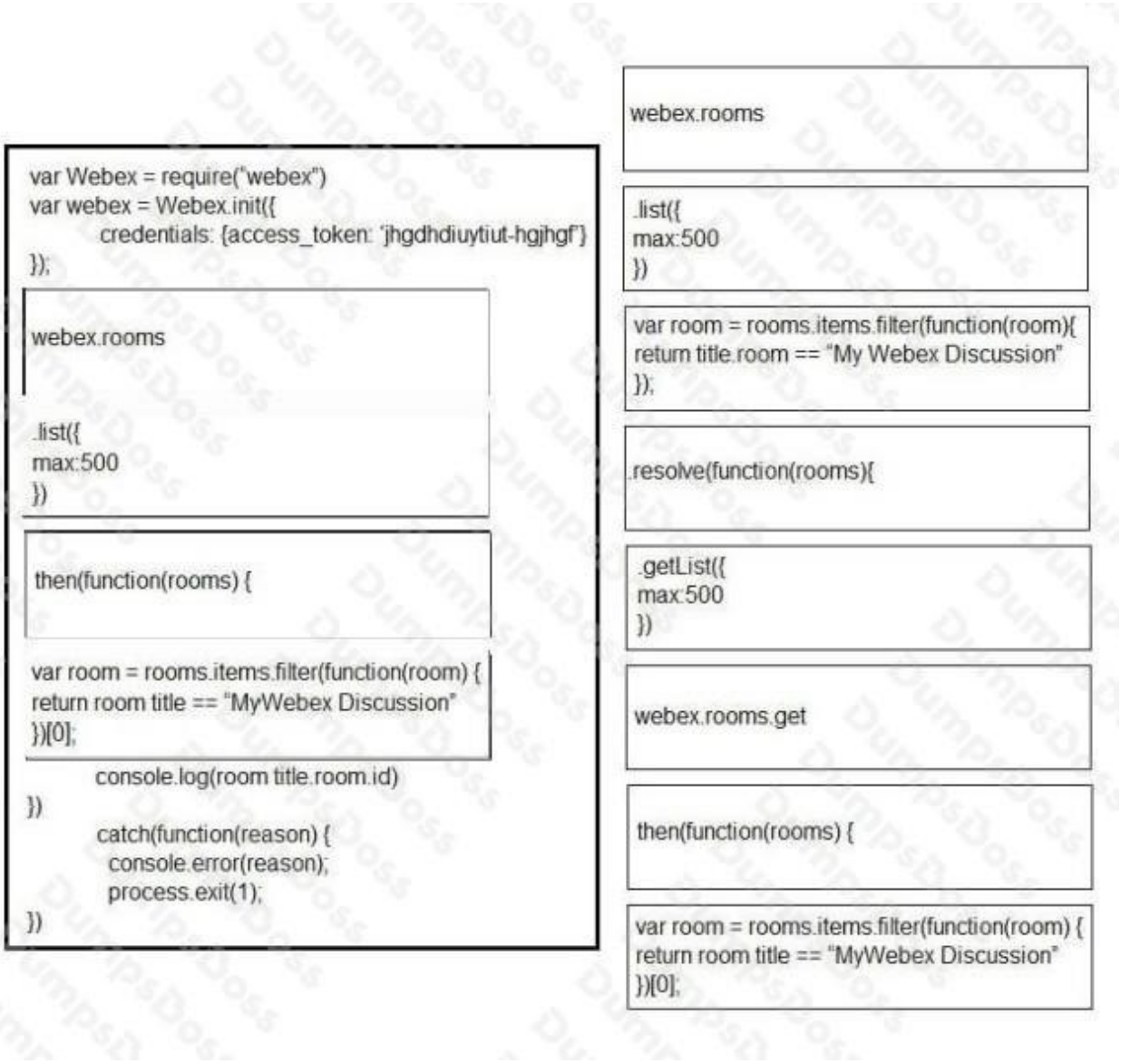
.getList({
max:500
})

webex.rooms.get

then(function(rooms) {

var room = rooms.items.filter(function(room) {
return room title == "MyWebex Discussion"
})[0];

**ANSWER:**




**Explanation:**

The completed script correctly uses the Webex JavaScript SDK's rooms resource as a promise-based API chain. After the SDK client is initialized with an access token, `webex.rooms` selects the Rooms API service exposed by that client. Calling `.list({ max:500 })` requests a collection of rooms and uses the supported maximum-result setting to retrieve up to 500 rooms in the response. The Rooms API returns the asynchronous result as a promise, so `.then(function(rooms) {` is the correct continuation point for working with the returned collection.

Inside that callback, the response's `items` array contains the individual room objects. Filtering that array with a callback that compares `room.title` to the requested title identifies rooms with that exact title. Applying `[0]` to the filtered array obtains the first matching room object, which makes the existing `console.log(room.title, room.id)` statement valid: both the human-readable room title and the unique Webex room ID are properties of the selected room. The fixed closing braces preserve the promise chain, and the existing `catch` handler remains attached to report a failed request or processing error. This is the intended pattern for listing Webex rooms, selecting a matching object from the returned items, and using its metadata. Cisco's Webex JavaScript SDK project documents the client SDK and its resource-based API usage in the Webex JavaScript SDK repository, while the Webex developer documentation describes the Rooms resource and room-object fields at List Rooms.

## QUESTION NO: 7

Which two statements about RoomOS macros are true? (Choose two.)

**A.** RoomOS macros are written in JavaScript.

**B.** A macro can use xAPI to issue commands and receive feedback.

**C.** RoomOS macros must be written in Python.

**D.** A macro can run only while an SSH client is connected to the device.

**E.** A macro is stored and executed exclusively in the Webex cloud.

**ANSWER: A B**

**Explanation:**

RoomOS macros are written in JavaScript and execute locally on the collaboration device. This allows a macro to implement device-specific automation, such as reacting to user-interface events, controlling peripherals, or applying local workflow logic.

A macro can use the xAPI JavaScript module to issue commands, query device state, and subscribe to status or event feedback. For example, a macro can subscribe to a user-interface event and then invoke an xAPI command in response. Macros do not require an external application to maintain an SSH connection while they run.

See the Macros on RoomOS documentation and the RoomOS xAPI documentation.

## QUESTION NO: 8

What is the primary purpose of the `state` parameter in the Webex OAuth authorization-code flow?

**A.** To encrypt the OAuth access token before it is stored.

**B.** To correlate the authorization response with the original request.

**C.** To identify the Webex organization that owns the integration.

**D.** To replace the client secret in a token request.

**ANSWER: B**

**Explanation:**

The **`state`** **parameter correlates the OAuth authorization response with the authorization request and helps protect the application against cross-site request forgery. Before redirecting the user to Webex, the application generates an unpredictable value, stores it in the user session, and includes it in the authorization request. When Webex redirects the user to the configured redirect URI, the application verifies that the returned value matches the stored value before exchanging the authorization code for tokens.**

**The `state` value is not an access token, client secret, or redirect URI. See the Webex Integrations documentation for the OAuth authorization flow.**

## QUESTION NO: 9

**Which two tasks should use a Webex bot instead of aWebex integration? (Choose two.)**

**A. Ask a QUESTION NO: in natural language to obtain the current price of a stock.**

**B. Translate a word or phrase from Italian to English.**

**C. Notify all users who send a 1:1 message that the user is currently out of the office.**

**D. Automatically delete a message that was sent with spelling or grammatical errors.**

**E. Archive all the messages in a group space.**

**ANSWER: A B**

**Explanation:**

**“Ask a QUESTION NO: in natural language to obtain the current price of a stock.” and “Translate a word or phrase from Italian to English.” are well suited to a Webex bot because each is a conversational, request-and-response workflow. A user can add the bot to a space or communicate with it directly, submit a natural-language request, and receive the requested result as a message. This model makes the bot a visible participant in the conversation and provides an intuitive chat-based interface for on-demand services such as market-data lookup and language translation.**

**Webex bots are application-controlled Webex identities that can receive messages in spaces where they participate and send responses, including rich interactive content when appropriate. They are intended to bring external services into Webex conversations without requiring every user to navigate away from the messaging workflow. A stock-price service can parse the user’s request, retrieve current market data from its backend or an external provider, and post a response. Likewise, a translation service can accept text, process it through a translation engine, and return the English result in the same conversation. Cisco’s bot documentation describes bots as programs that interact with users through Webex messaging spaces: Webex Bots. For the broader distinction between bot and user-authorized application patterns, see Webex Integrations.**